File and File Reader

A file object inherits from Blob and is extended with filesystem related capabilities

In addition to `Blob` methods and properties, `File` objects also have `name` and `lastModified` properties, plus the internal ability to read from filesystem. We usually get `File` objects from user input, like `<input>` or Drag'n'Drop events (`ondragend`).

FileReader is an object with the sole purpose of reading data from `Blob` (and hence `File` too) objects.

It delivers the data using events, as reading from disk may take time.

The constructor:

```
let reader = new FileReader(); // no arguments
```
The main methods:

- **`readAsArrayBuffer(blob)`** – read the data in binary format `ArrayBuffer`.
- **`readAsText(blob, [encoding])`** – read the data as a text string with the given encoding (`utf-8` by default).
- **`readAsDataURL(blob)`** – read the binary data and encode it as base64 data url.
- **`abort()`** – cancel the operation.

The choice of `read*` method depends on which format we prefer, how we're going to use the data.

- `readAsArrayBuffer` – for binary files, to do low-level binary operations. For high-level operations, like slicing, `File` inherits from `Blob`, so we can call them directly, without reading.
- `readAsText` – for text files, when we'd like to get a string.
- `readAsDataURL` – when we'd like to use this data in `src` for `img` or another tag. There's an alternative to reading a file for that, as discussed in chapter Blob: `URL.createObjectURL(file)`.

As the reading proceeds, there are events:

- `loadstart` – loading started.
- `progress` – occurs during reading.
- `load` – no errors, reading complete.
- `abort` – `abort()` called.
- `error` – error has occurred.
- `loadend` – reading finished with either success or failure.

When the reading is finished, we can access the result as:

- `reader.result` is the result (if successful)
- `reader.error` is the error (if failed).

The Fetch

The `fetch()` **method** is used to make asynchronous requests to the server and load the information that is returned by the server onto the web pages.

The Fetch API is a modern interface that allows you to make HTTP requests to servers from web browsers.

In addition, the Fetch API is much simpler and cleaner.

The fetch() method is available in the global scope that instructs the web browsers to send a request to a URL.

```
fetch('https://example.com/data.json')
 .then(response => response.json())
 .then(data => console.log(data));
```

Post request

HTTP stands for Hypertext Transfer Protocol, and it is a set of rules for transferring data between the client (browser) and the server. There are various HTTP methods, such as GET, POST, PUT, DELETE, and more. Among these, POST is one of the most commonly used methods. It is typically used to submit data to a server for processing, updating, or creating new resources.

Sending an image

To send an image use the **`fetch()`** API or other methods to make an HTTP POST request and include the image data in the request body